C# 프로그래밍

- 14 주차 (1강)

SEJONG UNIVERSITY

예외 처리

- 기본/고급 예외 처리
- System.Exception 클래스
- 예외 던지기
- finally 문
- 사용자 정의 예외 클래스

예외 (Exception)

- 개발자가 생각하는 시나리오에서 벗어나는 사건을 예외라고 함
  - ✓ 예를 들어, 배열의 범위 밖의 배열의 요소를 접근하려고 시도

```
static void Main(string[] args)
{
    int[] arr = { 1, 2, 3 };

    for(int i=0;i<5;i++)
    {
        Console.WriteLine(arr[i]);
    }
    Console.WriteLine("종료"); // 실행 X
}
```

```
출력 결과:
1
2
3
Unhandled exception. System.IndexOutOfRangeException:
Index was outside the bounds of the array.
   at KillingProgram.MainApp.Main(String[] args) in
C:₩Users₩ongte₩source₩repos₩KillingProgram₩KillingPr
ogram₩MainApp.cs:line 13
```

- 예제코드 for 문에서 배열 범위 밖의 요소 접근 시, 예외 메시지 출력하고 프로그램 종료
  - ✓ 배열 객체는 예외에 대한 상세정보를 IndexOutofRangeException 객체에 담은 후 Main() 메소드에 던짐
  - ✓ Main() 메소드는 예외를 CLR에 던짐
  - ✓ CLR까지 전달된 예외는 "처리되지 않은 예외"가 되고, 예외 관련 메시지 출력 후 강제 종료

예외 처리 (Exception Handling)

- 예외가 프로그램의 오류 또는 다운으로 이어지지 않도록 적절하게 처리
  - ✓ 기본 에러 처리와 고급 에러 처리로 나눌 수 있음
- 기본 예외 처리
  - ✓ 예외가 발생하지 않게 사전에 해결

```
static void Main(string[] args)
{
    int[] arr = { 1, 2, 3 };

    for(int i=0;i<5;i++)
    {
        // 기본 에러 처리
        if (i<arr.Length) // 인덱스가 배열의 길이를 넘는지 사전에 확인
            Console.WriteLine(arr[i]);
        else
            Console.WriteLine("인덱스 범위를 넘었습니다.");
    }
    Console.WriteLine("종료");
}
```

출력 결과:
1
2
3
인덱스 범위를 넘었습니다.
인덱스 범위를 넘었습니다.
종료

고급 에러 처리 : try ~ catch로 예외 받기

- C#에서 예외를 받을 때 try ~ catch 문을 이용
  - ✓ 이전 슬라이드 예제에서 배열이 IndexOutRangeException 예외를 던졌을 때, Main() 메소드는 try ~ catch 문으로 예외를 받을 수 있음
- 실행코드를 try 블록에 작성하고, try 블록에서 던지는 예외는 catch 블록에서 받음
  - ✓ catch 문은 try 블록에서 던지는 예외 객체와 형식이 일치해야 받을 수 있음
  - ✓ 모든 catch 문에서 예외를 받지 못하면 "처리되지 않은 예외"로 남게 됨

사용형식:

```
try
{
    // 실행하고자 하는 코드
}
catch(예외_객체_1)
{
    // 예외가 발생했을 때의 처리
}
catch(예외_객체_2)
{
    // 예외가 발생했을 때의 처리
}
```

고급 에러 처리 : try ~ catch 예제 코드

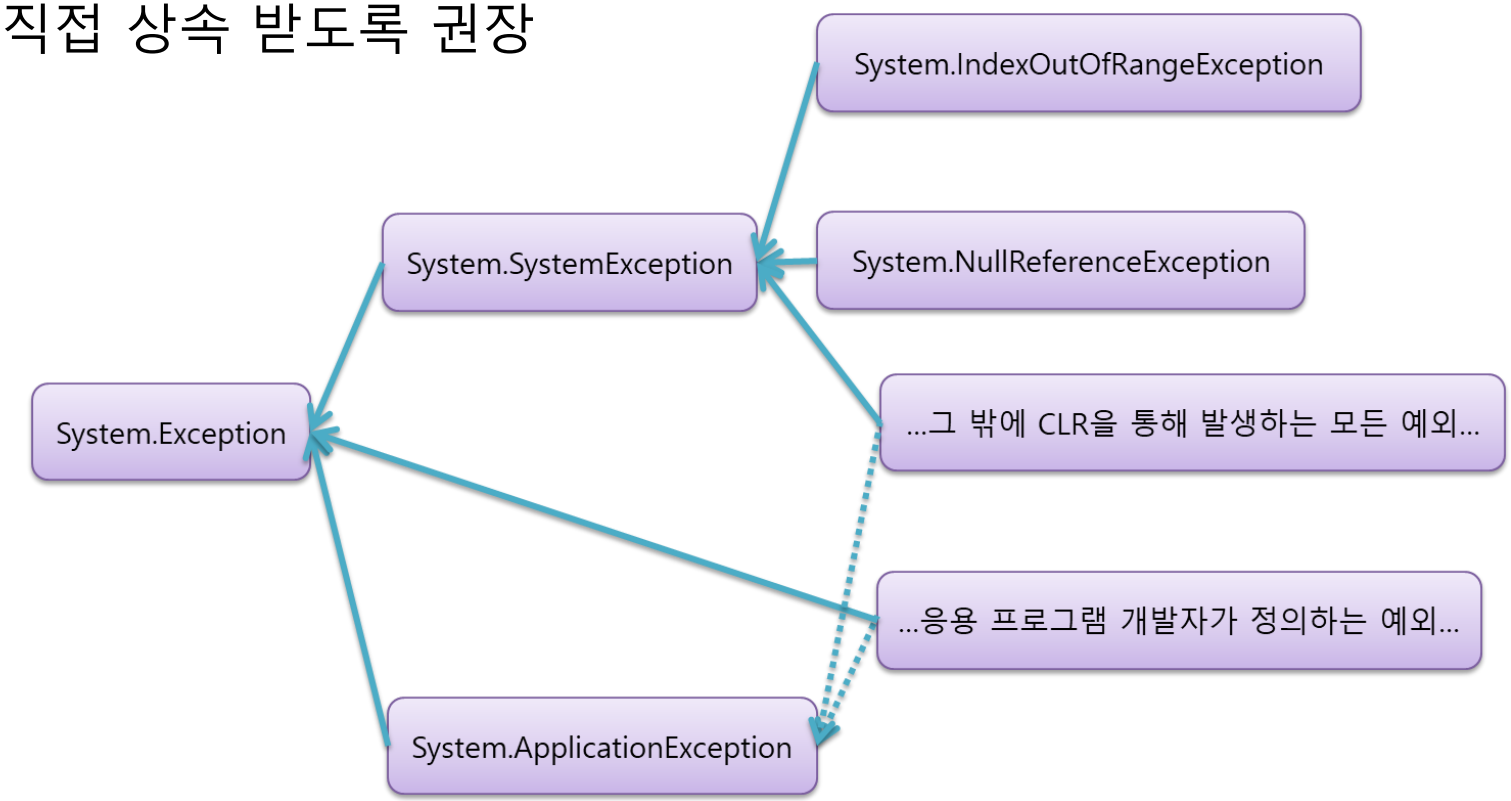
```
static void Main(string[] args)
{
    int[] arr = { 1, 2, 3 };

    try
    {
        for (int i = 0; i < 5; i++)
        {
            Console.WriteLine(arr[i]);
        }
        Console.WriteLine("정상적 수행완료");
    }
    catch(IndexOutOfRangeException e) // catch(IndexOutOfRangeException)와 같이 예외 타입만 작성도 가능
    {
        Console.WriteLine($"예외가 발생했습니다 : {e.Message}");
    }

    Console.WriteLine("종료");
}
```

출력 결과:

```
1
2
3
예외가 발생했습니다 : Index was outside the bounds of the array.
종료
```

System.Exception 클래스

- C#에서 모든 예외 클래스는 System.Exception 클래스를 상속 받음
  - ✓ IndexOutRangeException 예외도 System.Exception으로부터 파생
  - ✓ 관례상으로는 응용 프로그램 개발자 정의하는 예외는 ApplicationException 타입을 상속하고, CLR에서 미리 정의된 예외는 SystemException 타입을 상속
  - ✓ 하지만, 최근 닷넷 가이드라인 문서는 응용프로그램 개발자가 만드는 예외를 System.Exception에서 직접 상속 받도록 권장



System.Exception 클래스

- 상속관계로 인해 모든 예외 클래스는 System.Exception 형식으로 변환 가능
  - ✓ System.Exception 형식의 예외를 받는 catch절 하나면 모든 예외를 받을 수 있음
- 하나의 System.Exception 타입 사용 시, 코드를 면밀히 검토 필요
  - ✓ 처리하지 않아야 할 예외까지 처리하는 일이 없도록 주의

여러 개의 catch 절로 처리하는 예외

```
try
{
}
catch(IndexOutOfRangeException e)
{
    // ...
}
catch (DivideByZeroException e)
{
    // ...
}
```

하나의 catch 절로 모든 예외 처리 가능

```
try
{
}
catch(Exception e)
{
    // ...
}
```

System.Exception 클래스의 주요 멤버

멤버	타입	설명
Message	프로퍼티	예외를 설명하는 메시지 반환
Source	프로퍼티	예외를 발생시킨 응용 프로그램의 이름을 반환
StackTrace	프로퍼티	예외가 발생된 메서드의 호출 스택을 반환
ToString	메서드	Message, StackTrace 내용을 포함하는 문자열을 반환

```
static void Main(string[] args)
{
    int[] arr = { 1, 2, 3 };
    try
    {
        for (int i = 0; i < 5; i++)
        {
            Console.WriteLine(arr[i]);
        }
        Console.WriteLine("정상적 수행완료");
    }
    catch (IndexOutOfRangeException e)
    {
        Console.WriteLine($"예외 Message : {e.Message}");
        Console.WriteLine($"예외 Source : {e.Source}");
        Console.WriteLine($"예외 StackTrace : {e.StackTrace}");
        Console.WriteLine($"예외 ToString : {e.ToString()}");
    }
    Console.WriteLine("종료");
}
```

출력 결과:

```
1
2
3
예외 Message : Index was outside the bounds of the array.
예외 Source : ExceptionClass
예외 StackTrace :    at ExceptionClass.MainApp.Main(String[] args) in C:₩Users₩ongte₩source₩repos₩KillingProgram₩ExceptionClass₩MainApp.cs:line 15
예외 ToString : System.IndexOutOfRangeException: Index was outside the bounds of the array.
   at ExceptionClass.MainApp.Main(String[] args) in C:₩Users₩ongte₩source₩repos₩KillingProgram₩ExceptionClass₩MainApp.cs:line 15
종료
```

예외 던지기 : throw 문

- 예외 객체는 throw 문을 통해 던지고,
  - ✓ 던져진 예외는 try~catch 문을 통해 받을 수 있음

사용예제 1:

```
static void Main(string[] args)
{
    try
    {
        // ...
        throw new Exception("예외를 던집니다.");
    }
    catch(Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

throw 문을 통해 던져진 예외 객체는 catch 문을 통해 받음

사용예제 2: 메서드 안에서 던져진 예외를 메서드를 호출하는 try~catch 문에서 받음

```
static void Dosomething(int arg)
{
    if (arg < 10)
        Console.WriteLine( "arg : {0}" , arg);
    else
        throw new Exception( "arg가 10보다 큽니다." );
}

Static void Main(string[] args)
{
    try
    {
        Dosomething(13);
    }
    catch(Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

이 예외는 DoSomething() 메서드의 호출자에게 던져 짐

DoSomething() 메서드에서 던진 예외는 호출자의 try~catch 문에서 받음

출력 결과:
arg가 10보다 큽니다.

예외 던지기 : throw 식

- C# 7.0부터는 throw를 식으로도 사용할 수 있도록 지원

사용예제 1: null 병합 연산자 (??)

```
int? a = null;
int b = a ?? throw new ArgumentNullException();
```

a는 null이므로, b에 a를 할당하지 않고 throw 문을 실행

사용예제 2: 조건 연산자

```
int[] array = new[] { 1,2,3};
int index = 4;
int value = array[
    index >= 0 && index < 3
    ? index : throw new IndexOutOfRangeException()
    ];
```

Index는 3보다 크기 때문에 IndexOutOfRangeException 예외를 던짐

예외 던지기 : throw 식 예제 코드

```
static void Main(string[] args)
{
    try {
        int? a = null;
        int b = a ?? throw new ArgumentNullException();
    }
    catch(ArgumentNullException e)
    {
        Console.WriteLine(e.Message);
    }

    try {
        int[] array = new[] { 1, 2, 3 };
        int index = 4;
        int value = array[
            index >= 0 && index < 3
            ? index : throw new IndexOutOfRangeException()
            ];
    }
    catch (IndexOutOfRangeException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

출력 결과:

```
Value cannot be null.
Index was outside the bounds of the array.
```

고급 에러 처리 : try ~ catch 와 finally

- try 블록 코드 실행 중 에러가 던져지면 catch 절로 바로 뛰어넘어 옴
  - ✓ 만약 try 블록의 자원 해제 같은 중요한 코드를 미처 실행하지 못한다면 문제가 됨
- try ~ catch문의 마지막에 연결해서 사용하는 finally 절
  - ✓ try 절이 실행된다면 finally 절은 어떤 경우라도 실행 됨
  - ✓ 예외 처리 시, 자원 해제하는 코드를 넣어두는 용도로 적합

finally 절이 없는 경우: try 와 catch 블록 모두 파일 닫는 코드 포함

```
FileStream file = null;

try
{
    file = ... [파일열기]...;
    // .. 열린 파일로 작업, 이 과정에서 예외 발생할 수 있음
    file.Close();
}
catch (XXXException e)
{
    // ...
    file.Close();
}
```

finally 절에만 파일을 닫는 코드를 포함하여 처리

```
FileStream file = null;

try
{
    file = ... [파일열기]...;
    // .. 열린 파일로 작업, 이 과정에서 예외 발생할 수 있음
}
catch (XXXException e)
{
    // ...
}
finally
{
    file.Close();
}
```

고급 에러 처리 : try ~ catch 와 finally

- try 절 안에 return 문이나 throw 문이 사용되어도 finally 절은 반드시 실행
  - ✓ 단, finally 절 안에 return 문 사용은 컴파일 에러 발생

```
static int Divide(int dividend, int divisor)
{
    try
    {
        Console.WriteLine("Divide() 시작");
        return dividend / divisor; // 예외가 일어나지 않고 정상적으로 return 해도 finally 절 실행
    }
    catch(DivideByZeroException e)
    {
        Console.WriteLine("Divide() 예외 발생");
        throw e; // 예외가 일어나더라도 finally 절은 실행
    }
    finally
    {
        Console.WriteLine("Divide() 끝");
    }
}
```

고급 에러 처리 : try ~ catch 와 finally 예제 코드

```
static int Divide(int dividend, int divisor)
{
    try {
        Console.WriteLine("Divide() 시작");
        return dividend / divisor;
    }
    catch(DivideByZeroException e) {
        Console.WriteLine("Divide() 예외 발생");
        throw e;
    }
    finally {
        Console.WriteLine("Divide() 끝");
    }
}
```

```
static void Main(string[] args)
{
    try {
        Console.Write("제수를 입력하세요. :");
        string temp = Console.ReadLine();
        int dividend = Convert.ToInt32(temp);

        Console.Write("피제수를 입력하세요. : ");
        temp = Console.ReadLine();
        int divisor = Convert.ToInt32(temp);

        Console.WriteLine("{0}/{1} = {2}",
            dividend, divisor, Divide(dividend, divisor));
    }
    catch (FormatException e) {
        Console.WriteLine("에러 : " + e.Message);
    }
    catch (DivideByZeroException e) {
        Console.WriteLine("에러 : " + e.Message);
    }
    finally {
        Console.WriteLine("프로그램을 종료합니다.");
    }
}
```

출력 결과:

```
제수를 입력하세요. :40
피제수를 입력하세요. : 십일
에러 : Input string was not in a correct
format.
프로그램을 종료합니다.

제수를 입력하세요. :7
피제수를 입력하세요. : 0
Divide() 시작
Divide() 예외 발생
Divide() 끝
에러 : Attempted to divide by zero.
프로그램을 종료합니다.
```

사용자 정의 예외 클래스

- 개발자는 C#에서 정의 되어 있지 않은 새로운 예외 클래스를 만들 수 있음
  - ✓ 반드시 Exception 클래스를 상속하는 파생 클래스

사용 형식 :

```
class MyException : Exception
{
    // ...
}
```

- 사용자 정의 예외 예제 프로그램 (Next 슬라이드에 코드 설명)
  - ✓ 8비트 정수 (색을 구성하는 Alpha, Red, Green, Blue 값)를 매개변수로 입력 받아 32비트 정수 안에 병합하는 MergeARGB() 메소드 구현
  - ✓ 매개변수 입력 값이 0~255 사이이고, 이 범위를 벗어나면 InvalidArgumentException 예외 발생

예제 코드

```
class InvalidArgumentException : Exception
{
    public InvalidArgumentException() { }
    public object Argument { get; set; }
    public string Range { get; set; }
}
```

출력 결과:

```
0xFF000000
0x1000000
Exception of type
'MyException.InvalidArgumentException' was thrown.
Argument:300, Range:0~255
```

```
class MainApp
{
    static uint MergeARGB(uint alpha, uint red, uint green, uint blue)
    {
        uint[] args = new uint[] { alpha, red, green, blue };
        foreach(uint arg in args) {
            if (arg > 255)
                throw new InvalidArgumentException() {
                    Argument = arg,
                    Range = "0~255 “ };
        }
        return (alpha << 24 & 0xFF000000) |
            (red << 16 & 0xFF000000) |
            (green << 8 & 0xFF000000) |
            (blue & 0xFF000000);
    }
    static void Main(string[] args)
    {
        try {
            Console.WriteLine("0x{0:X}", MergeARGB(255,111,111,111));
            Console.WriteLine("0x{0:X}", MergeARGB(1, 65, 192, 128));
            Console.WriteLine("0x{0:X}", MergeARGB(0, 255, 255, 300));
        }
        catch(InvalidArgumentException e) {
            Console.WriteLine(e.Message);
            Console.WriteLine($"Argument:{e.Argument}, Range:{e.Range}");
        }
    }
}
```

Thank you!